Stan User Guide Models

Chris Waller

2020-10-11

Regression Models

Linear Regression

```
x_1_1 = cbind(runif(n_samples), runif(n_samples))
alpha_1_1 = 0.3
beta_1_1 = c(1.2, 1.6)
sigma_1_1 = 0.2
y_1_1 = alpha_1_1 + c(x_1_1 %*% beta_1_1) + rnorm(n_samples, 0, sigma_1_1)

linear_reg_data =
  list(
    N = n_samples,
    K = length(beta_1_1),
    y = y_1_1,
    x = x_1_1
  )
```

```
data {
  int<lower=0> N;   // number of data items
  int<lower=0> K;   // number of predictors
  matrix[N, K] x;   // predictor matrix
  vector[N] y;      // outcome vector
}
parameters {
  real alpha;           // intercept
  vector[K] beta;       // coefficients for predictors
  real<lower=0> sigma;  // error scale
}
model {
  y ~ normal(alpha + x * beta, sigma);  // likelihood
}
```

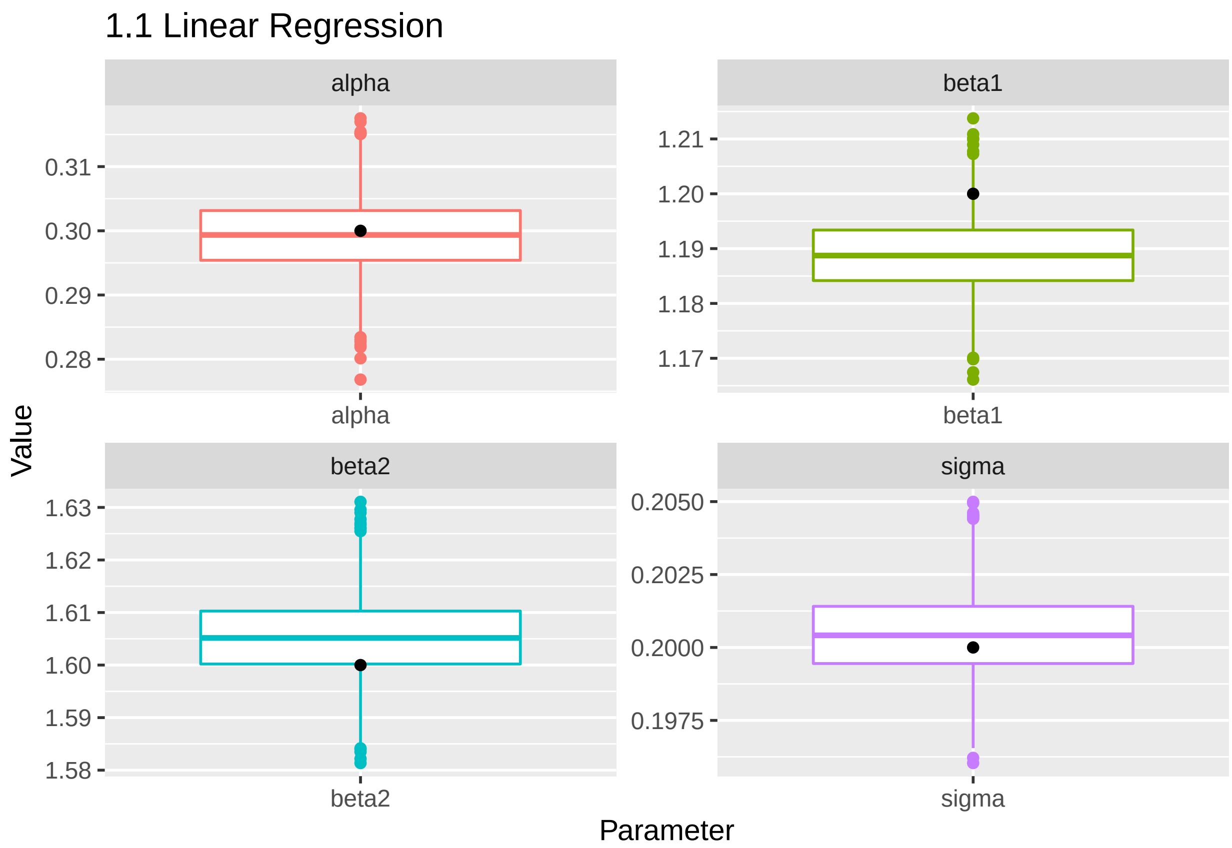
```
linear_reg_then = Sys.time()

linear_reg =
  sampling(
    linear_reg_model,
    data = linear_reg_data,
    iter = 1000,
    chains = 3,
    refresh = 50,
```

```
    warmup = 250
  )

linear_reg_time = difftime(Sys.time(), linear_reg_then)

linear_reg_time
```

```
## Time difference of 5.903346 secs
```

```
linear_reg_out = rstan::extract(linear_reg)

linear_reg_avg = lapply(linear_reg_out, function(x) colMeans(as.matrix(x)))

linear_reg_ggdata =
  data.frame(
    beta1 = linear_reg_out$beta[,1],
    beta2 = linear_reg_out$beta[,2],
    alpha = linear_reg_out$alpha,
    sigma = linear_reg_out$sigma
  ) %>%
  pivot_longer(
    cols = everything(),
    names_to = "param",
    values_to = "value"
  )

linear_reg_act =
  data.frame(
    param = c("beta1", "beta2", "alpha", "sigma"),
    value = c(beta_1_1, alpha_1_1, sigma_1_1)
  )

ggplot() +
  geom_boxplot(
    data = linear_reg_ggdata,
    aes(
      x = factor(param),
      y = value,
      color = param
      )
    ) +
  geom_point(
    data = linear_reg_act,
    aes(
      x = factor(param),
      y = value
    )
  ) +
  theme(
    legend.position = "none"
  ) +
  labs(
    title = "1.1 Linear Regression",
    y = "Value",
```

```
    x = "Parameter"
  ) +
  facet_wrap(
    ~ param,
    scales = "free"
  )
```



The QR Reparameterization

```
x_1_2 = cbind(runif(n_samples), runif(n_samples))
alpha_1_2 = 0.3
beta_1_2 = c(1.2, 1.6)
sigma_1_2 = 0.2
y_1_2 = alpha_1_2 + c(x_1_2 %*% beta_1_2) + rnorm(n_samples, 0, sigma_1_2)

qr_reparam_data =
  list(
    N = n_samples,
    K = length(beta_1_2),
    y = y_1_2,
    x = x_1_2
  )
```

```
data {
  int<lower=0> N;    // number of data items
```

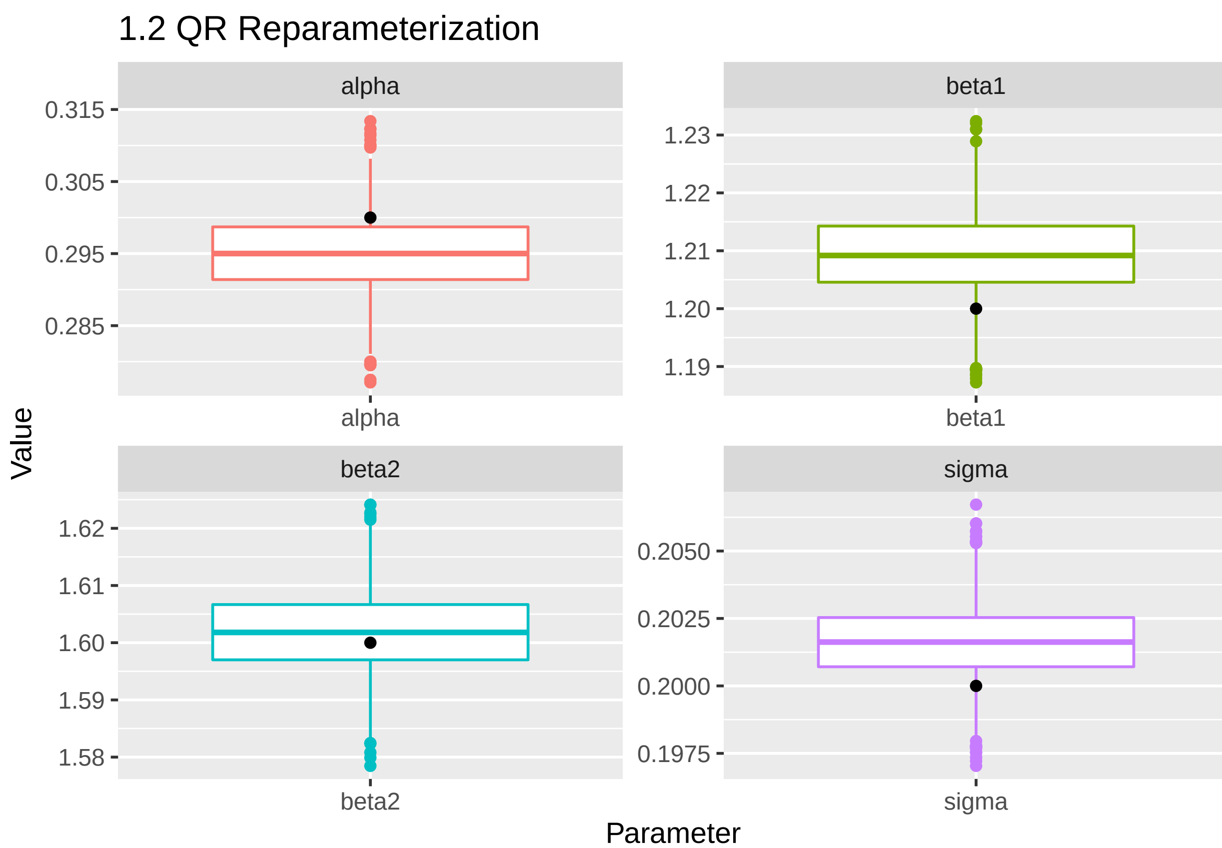
```
  int<lower=0> K;   // number of predictors
  matrix[N, K] x;   // predictor matrix
  vector[N] y;      // outcome vector
}
transformed data {
  matrix[N, K] Q_ast;
  matrix[K, K] R_ast;
  matrix[K, K] R_ast_inverse;
  // thin and scale the QR decomposition
  Q_ast = qr_Q(x)[, 1:K] * sqrt(N - 1);
  R_ast = qr_R(x)[1:K, ] / sqrt(N - 1);
  R_ast_inverse = inverse(R_ast);
}
parameters {
  real alpha;           // intercept
  vector[K] theta;      // coefficients on Q_ast
  real<lower=0> sigma;  // error scale
}
model {
  y ~ normal(Q_ast * theta + alpha, sigma);  // likelihood
}
generated quantities {
  vector[K] beta;
  beta = R_ast_inverse * theta; // coefficients on x
}
```

```r
qr_reparam_then = Sys.time()

qr_reparam =
  sampling(
    qr_reparam_model,
    data = qr_reparam_data,
    iter = 1000,
    chains = 3,
    refresh = 50,
    warmup = 250
  )

qr_reparam_time = difftime(Sys.time(), qr_reparam_then)

qr_reparam_time
```

```
## Time difference of 7.41499 secs
```

```r
qr_reparam_out = rstan::extract(qr_reparam)

qr_reparam_avg = lapply(qr_reparam_out, function(x) colMeans(as.matrix(x)))

qr_reparam_ggdata =
  data.frame(
    beta1 = qr_reparam_out$beta[,1],
    beta2 = qr_reparam_out$beta[,2],
    alpha = qr_reparam_out$alpha,
    sigma = qr_reparam_out$sigma
  ) %>%
```

```
  pivot_longer(
    cols = everything(),
    names_to = "param",
    values_to = "value"
  )

qr_reparam_act =
  data.frame(
    param = c("beta1", "beta2", "alpha", "sigma"),
    value = c(beta_1_2, alpha_1_2, sigma_1_2)
  )

ggplot() +
  geom_boxplot(
    data = qr_reparam_ggdata,
    aes(
      x = factor(param),
      y = value,
      color = param
    )
  ) +
  geom_point(
    data = qr_reparam_act,
    aes(
      x = factor(param),
      y = value
    )
  ) +
  theme(
    legend.position = "none"
  ) +
  labs(
    title = "1.2 QR Reparameterization",
    y = "Value",
    x = "Parameter"
  ) +
  facet_wrap(
    ~ param,
    scales = "free"
  )
```



Priors for Coefficients and Scales

No model in this section.

Robust Noise Models

```
x_1_4 = cbind(runif(n_samples), runif(n_samples))
alpha_1_4 = 2
beta_1_4 = c(7, 9)
sigma_1_4 = 1.5
nu_degf = 5 # Degrees of freedom of t-distribution
y_1_4 = alpha_1_4 + c(x_1_4 %*% beta_1_4) + rt(n_samples, nu_degf) * sigma_1_4

robust_noise_data =
  list(
    N  = n_samples,
    K  = length(beta_1_4),
    y  = y_1_4,
    x  = x_1_4,
    nu = nu_degf
  )
```

```
data {
  int<lower=0> N;   // number of data items
  int<lower=0> K;   // number of predictors
```

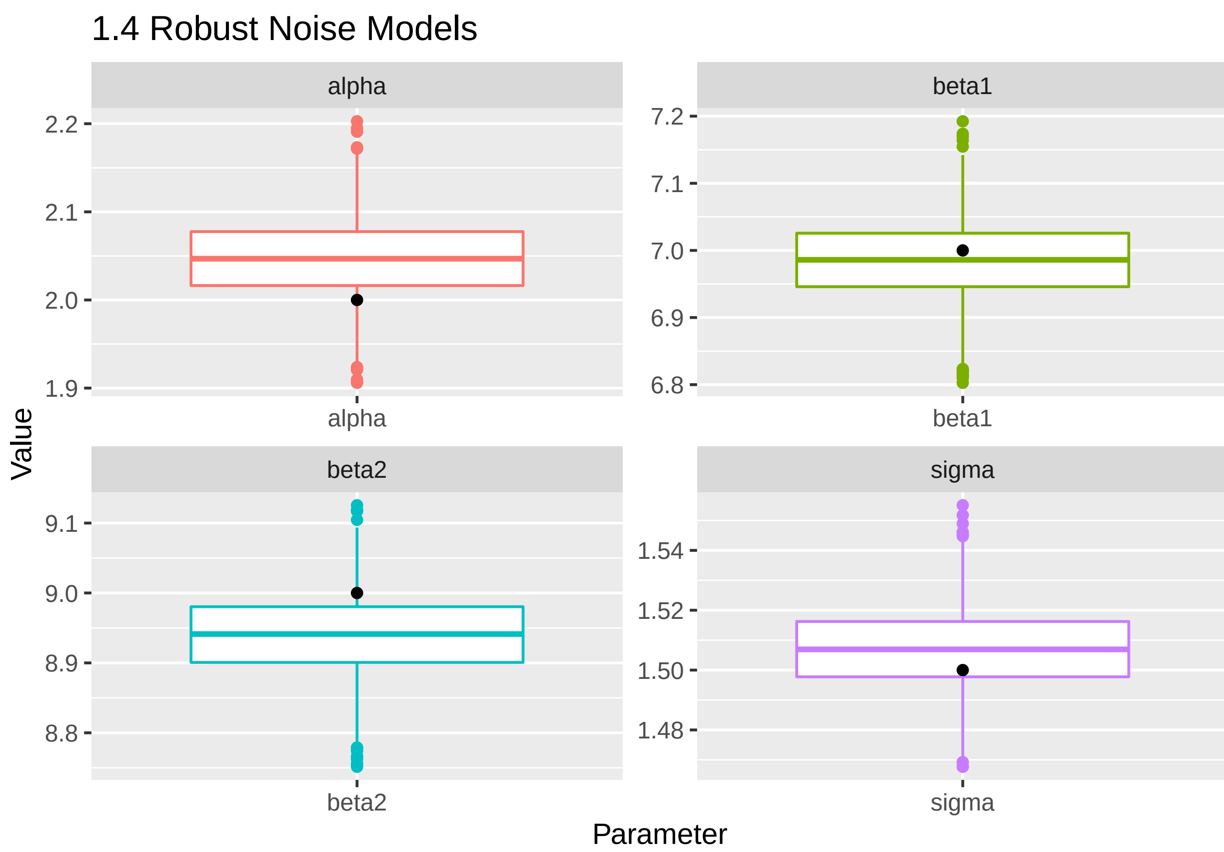
```
  matrix[N, K] x;   // predictor matrix
  vector[N] y;      // outcome vector
  real<lower=0> nu; // Noise Degrees of freedom
}
transformed data {
  matrix[N, K] Q_ast;
  matrix[K, K] R_ast;
  matrix[K, K] R_ast_inverse;
  // thin and scale the QR decomposition
  Q_ast = qr_Q(x)[, 1:K] * sqrt(N - 1);
  R_ast = qr_R(x)[1:K, ] / sqrt(N - 1);
  R_ast_inverse = inverse(R_ast);
}
parameters {
  real alpha;           // intercept
  vector[K] theta;      // coefficients on Q_ast
  real<lower=0> sigma;  // error scale
}
model {
  y ~ student_t(nu, Q_ast * theta + alpha, sigma);  // likelihood
}
generated quantities {
  vector[K] beta;
  beta = R_ast_inverse * theta; // coefficients on x
}
```

```r
robust_noise_then = Sys.time()

robust_noise =
  sampling(
    robust_noise_model,
    data = robust_noise_data,
    iter = 1000,
    chains = 3,
    refresh = 50,
    warmup = 250
  )

robust_noise_time = difftime(Sys.time(), robust_noise_then)

robust_noise_time
```

```
## Time difference of 11.97724 secs
```

```r
robust_noise_out = rstan::extract(robust_noise)

robust_noise_avg = lapply(robust_noise_out, function(x) colMeans(as.matrix(x)))

robust_noise_ggdata =
  data.frame(
    beta1 = robust_noise_out$beta[,1],
    beta2 = robust_noise_out$beta[,2],
    alpha = robust_noise_out$alpha,
    sigma = robust_noise_out$sigma
  ) %>%
```

```
  pivot_longer(
    cols = everything(),
    names_to = "param",
    values_to = "value"
  )

robust_noise_act =
  data.frame(
    param = c("beta1", "beta2", "alpha", "sigma"),
    value = c(beta_1_4, alpha_1_4, sigma_1_4)
  )

ggplot() +
  geom_boxplot(
    data = robust_noise_ggdata,
    aes(
      x = factor(param),
      y = value,
      color = param
    )
  ) +
  geom_point(
    data = robust_noise_act,
    aes(
      x = factor(param),
      y = value
    )
  ) +
  theme(
    legend.position = "none"
  ) +
  labs(
    title = "1.4 Robust Noise Models",
    y = "Value",
    x = "Parameter"
  ) +
  facet_wrap(
    ~ param,
    scales = "free"
  )
```



Logistic and Probit Regression

```
u_1_5 = runif(n_samples)
x_1_5 = runif(n_samples)
alpha_1_5 = 2
beta_1_5 = 0.1
logit_inv_1_5 = 1/(1 + exp(-(alpha_1_5 + beta_1_5 * x_1_5)))
y_1_5 = u_1_5 < logit_inv_1_5

log_and_pro_data =
  list(
    N = n_samples,
    x = x_1_5,
    y = y_1_5
  )
```

```
data {
  int<lower=0> N;
  vector[N] x;
  int<lower=0,upper=1> y[N];
}
parameters {
  real alpha;
  real beta;
}
```

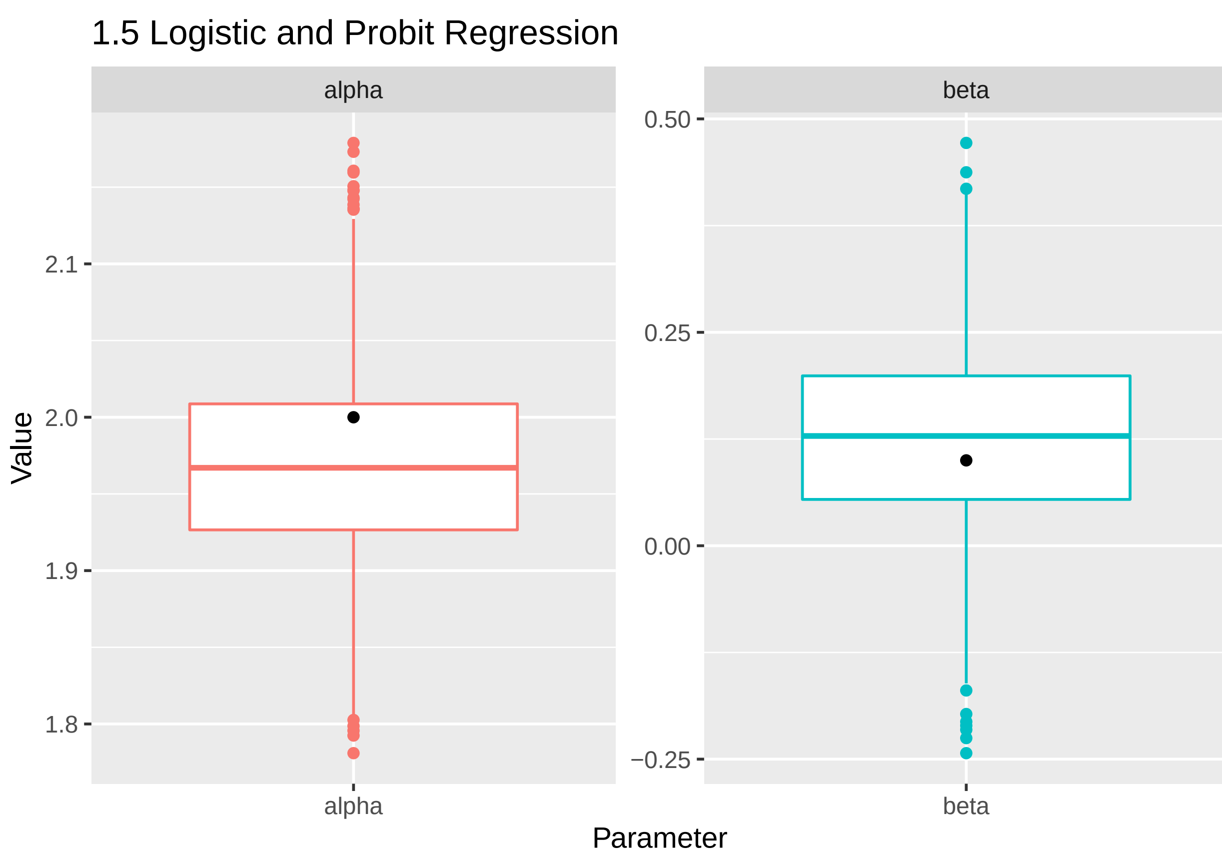
```
model {

  // Logistic
  y ~ bernoulli_logit(alpha + beta * x);

  // Probit
  // y ~ bernoulli(Phi(alpha + beta * x));

}
```

```r
log_and_pro_then = Sys.time()

log_and_pro =
  sampling(
    log_and_pro_model,
    data = log_and_pro_data,
    iter = 1000,
    chains = 3,
    refresh = 50,
    warmup = 250
  )

log_and_pro_time = difftime(Sys.time(), log_and_pro_then)

log_and_pro_time
```

```
## Time difference of 9.693312 secs
```

```r
log_and_pro_out = rstan::extract(log_and_pro)

log_and_pro_avg = lapply(log_and_pro_out, function(x) colMeans(as.matrix(x)))

log_and_pro_ggdata =
  data.frame(
    alpha = log_and_pro_out$alpha,
    beta = log_and_pro_out$beta
  ) %>%
  pivot_longer(
    cols = everything(),
    names_to = "param",
    values_to = "value"
  )

log_and_pro_act =
  data.frame(
    param = c("alpha", "beta"),
    value = c(alpha_1_5, beta_1_5)
  )

ggplot() +
  geom_boxplot(
    data = log_and_pro_ggdata,
    aes(
      x = factor(param),
      y = value,
```

```r
      color = param
    )
  ) +
  geom_point(
    data = log_and_pro_act,
    aes(
      x = factor(param),
      y = value
    )
  ) +
  theme(
    legend.position = "none"
  ) +
  labs(
    title = "1.5 Logistic and Probit Regression",
    y = "Value",
    x = "Parameter"
  ) +
  facet_wrap(
    ~ param,
    scales = "free"
  )
```



Multi-Logit Regression

```r
k_1_6 = 3 # Possible output variables for y
d_1_6 = 2
x_1_6 = matrix(runif(n_samples * d_1_6), n_samples, d_1_6)
beta_1_6 = matrix(c(2, 1.5, 1.4, 2, 1.7, 1.8), d_1_6, k_1_6)
x_beta_1_6 = x_1_6 %*% beta_1_6

softmax =  function(u) exp(u) / sum(exp(u))

sm_1_6 = t(apply(x_beta_1_6, 1, softmax))

y_1_6 = apply(sm_1_6, 1, which.max)

multi_logit_data =
  list(
    N = n_samples,
    D = d_1_6,
    K = k_1_6,
    x = x_1_6,
    y = y_1_6
  )
```

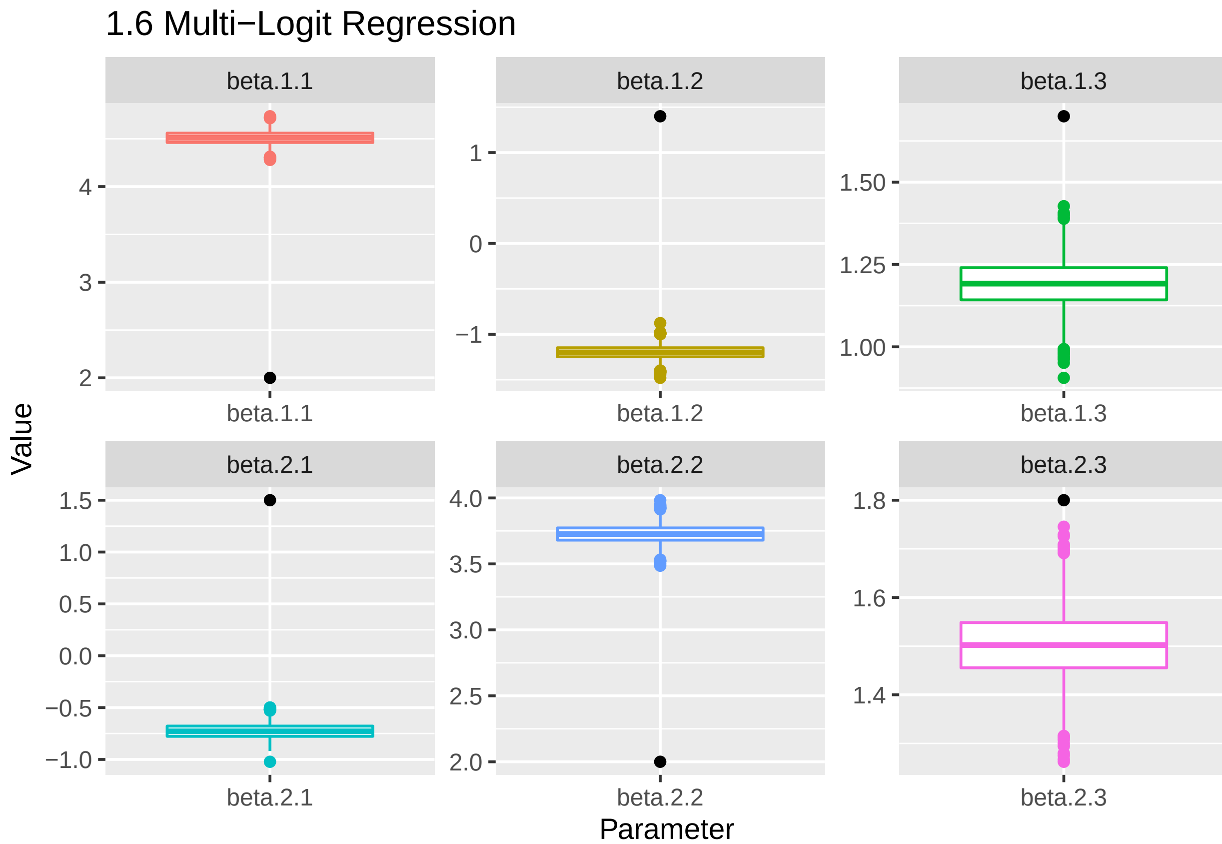
```stan
data {
  int K;
  int N;
  int D;
  int y[N];
  matrix[N, D] x;
}
parameters {
  matrix[D, K] beta;
}
model {
  matrix[N, K] x_beta = x * beta;

  to_vector(beta) ~ normal(1.5, 0.1);

  for (n in 1:N)
    y[n] ~ categorical_logit(x_beta[n]');
}
```

```r
multi_logit_then = Sys.time()

multi_logit =
  sampling(
    multi_logit_model,
    data = multi_logit_data,
    iter = 1000,
    chains = 3,
    refresh = 10,
    warmup = 250
  )
```

```
multi_logit_time = difftime(Sys.time(), multi_logit_then)

multi_logit_time
```

```
## Time difference of 1.128875 mins
```

```
multi_logit_out = rstan::extract(multi_logit)

multi_logit_avg = apply(multi_logit_out$beta, c(2, 3), mean)

multi_logit_ggdata =
  data.frame(
    beta = multi_logit_out$beta
  ) %>%
  pivot_longer(
    cols = everything(),
    names_to = "param",
    values_to = "value"
  )

multi_logit_act =
  data.frame(
    param = paste0("beta.", rep(seq(d_1_6), k_1_6), ".", rep(seq(k_1_6), rep(d_1_6, k_1_6))),
    value = c(beta_1_6)
  )

ggplot() +
  geom_boxplot(
    data = multi_logit_ggdata,
    aes(
      x = factor(param),
      y = value,
      color = param
    )
  ) +
  geom_point(
    data = multi_logit_act,
    aes(
      x = factor(param),
      y = value
    )
  ) +
  theme(
    legend.position = "none"
  ) +
  labs(
    title = "1.6 Multi-Logit Regression",
    y = "Value",
    x = "Parameter"
  ) +
  facet_wrap(
    ~ param,
    scales = "free"
  )
```



Parameterizing Centered Vectors